CamComSim: a LED-to-Camera Communication Simulator

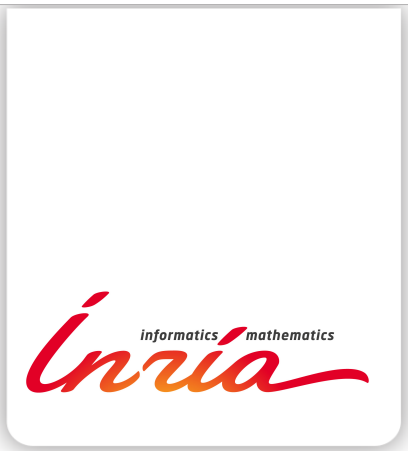
Alexis Duque, Razvan Stanica, Hervé Rivano, Adrien Desportes

CamComSim: a LED-to-Camera Communication Simulator

Alexis Duque , Razvan Stanica , Herve Rivano , Adrien Desportes

RESEARCH REPORT
N° 9114
October 2017
Project-Team AGORA

ISSN 0249-6399 ISRN INRIA/RR--9114--FR+ENG



CamComSim: a LED-to-Camera Communication Simulator

Alexis Duque * †, Razvan Stanica *, Herve Rivano *, Adrien Desportes †

Project-Team AGORA

Research Report n° 9114 — October 2017 — 15 pages

Abstract: In this paper, we present CamComSim, the first simulator for development and rapid prototyping of LED-to-Camera communication systems. Our event driven simulator relies on a standalone Java application that is easily extensible through a set of interfaces. A range of low and high-level parameters, such as the camera characteristics, the physical layer data unit size, or the redundancy mechanism can be chosen. CamComSim uses empirically validated models for the LED-to-Camera channel and the broadcast protocols, configurable with a finely grained precision. To validate CamComSim implementation and accuracy, we use a real life testbed based on a color LED and a smartphone and compare the performance reached by the testbed with the results given by our simulator. We illustrate with a real use case the full usage of CamComSim, tuning a broadcast protocol that implements the transmission of 1 kbyte of information. The results highlight that our simulator is very precise and predicts the performance of a real LED-to-Camera system with less than 10% of error in most cases.

Key-words: VLC, simulator, LED-to-Camera Communication, IoT, smartphone

* Univ Lyon, INSA Lyon, Inria, CITI, F-69621, Villeurbanne, France
† Rtone, Lyon, France

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

CamComSim: un simulateur de communication LED vers camera

Résumé : Dans cet article, nous présentons CamComSim, le premier simulateur de systèmes de communication par lumière visible d'une LED vers une caméra. Notre simulateur repose sur une application Java autonome qui est facilement extensible grâce un ensemble d'interfaces. De nombreux paramètres bas et haut niveau, tels que les caractéristiques de la caméra, la taille des trames PHY ou le mécanisme de redondance utilisé peuvent être choisis. CamComSim utilise des modèles de canal et de protocole de broadcast validés empiriquement, et finement configurable. Pour valider l'implémentation et la précision de CamComSim, nous utilisons un banc de test basé sur une LED couleur et un smartphone et comparons les performances obtenues par le banc de test avec les résultats fournis par notre simulateur. Nous illustrons avec un cas d'utilisation réel l'utilisation de CamComSim, en ajustant les paramètres d'un protocole de broadcast lors de la transmission de 1 Ko d'information. Les résultats mettent en évidence que notre simulateur est très précis et prédit les performances d'un vrai système LED vers caméra avec moins de 10

Mots-clés : communication par lumière visible, simulateur, smartphone, internet des objets, communication LED vers camera

1 Introduction

In 2012, *Danakis et al.* [1] demonstrated the first communication between a light emitting diode (LED) and a smartphone using the rolling shutter effect of the smartphone camera. Since then, and speeded up by the omnipresence of smartphones in the people daily life, LED-to-Camera Communication, also known as Optical Camera Communication (OCC), has become more and more attractive. An example is the recently created working group within the IEEE to incorporate OCC physical layer functionalities in the formerly established IEEE 802.15.7 visible light communication (VLC) standard [2].

In LED-to-camera VLC, the LED is modulating the light it emits using an on-off keying (OOK) scheme to encode information. On the smartphone side, the camera is used to take a picture of the transmitter LED, or of a surface illuminated by it. These two topologies are termed respectively line-of-sight (LOS) and non-line-of-sight (NLOS). As a consequence of the rolling shutter effect, the OOK modulation used by the LED will create a series of dark and illuminated stripes on the picture. These stripes encode the information transmitted by the LED.

Previous studies [3, 4] evaluate the performance of LED-to-camera communications by using commercial lighting LED bulbs or low-power color LED [5]. These works propose several protocols and mechanisms to efficiently broadcast information and face the inevitable packet losses in such systems. All these evaluations were experimental, using different testbeds, but none of them took advantage of preliminary simulations to avoid tedious experimentation campaigns conducted in order to design and improve the proposed communication protocols. Simulations are a useful tool to reproduce the behavior of a system, for performance analysis purposes, and widely used to gain insights into a technology. The simulation process often relies on theoretical and experimental models of the behavior of the system being studied. Simulations are extremely useful in testing different setups and configurations, offering valuable information which helps to optimize the system and its performance. Another benefit of the simulations is the possibility of reproducing certain conditions or scenarios in a repetitive way and in a time-efficient manner. From this point of view, simulations represent a necessary step towards the implementation and the testing of a system.

Nonetheless, no work related to LED-to-Camera communication simulation exists in the literature: in this paper, we propose CamComSim, the first LED-to-Camera communication simulator.

CamComSim is a standalone Java application that does not depend on an existing simulation framework. Therefore, CamComSim is fully configurable through a large set of settings and can be extended to comply with any LED-to-Camera system. We conduct a series of experiments on a testbed, demonstrating that CamComSim gives results close to the reality for a wide range of transmission parameters. Thus, we expect our contribution will help further efforts on the design of new LED-to-Camera communication protocols.

The rest of this paper is organized as follow. Sec. 2 presents previous works and spotlights the lack of simulation tools that motivates our work. We then study the LED-to-Camera communication properties and propose analytical models in Sec. 3, before detailing the CamComSim implementation in Sec. 4. Sec. 5 focuses on the simulator validation, confronting simulation results with the experimental performance reached by our testbed. Finally, Sec. 6 gives a practical use case for CamComSim, before the concluding remarks in Sec. 7.

2 Related Works

Network simulation has been largely studied in the field of wireless communication [6]. Nonetheless, VLC simulation remains poorly investigated and VLC simulation tools are still missing.

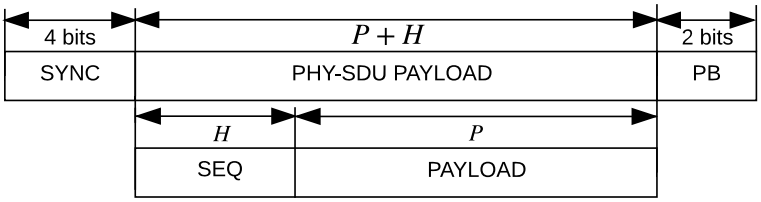


Figure 1: The PHY-SDU (top) and Packet (bottom) format. A Packet is a PHY-SDU payload with a sequence number and a payload. SYNC stand for synchronization symbol and PB are 2 parity bits.

The main efforts on simulating VLC systems have focused on the indoor channel simulation [7], or on the 802.15.7 PHY [8, 9] and MAC [10] layers. These approaches rely on classical network simulation frameworks such as the ones used in the context of wireless and ad hoc networks, e.g. NS-2 [10], NS-3 [8], OMNET++[9] or MATLAB [7].

Since all these works consider LED-to-Photodiode communication, LED-to-Camera communication remains completely unexplored. Our work is the first effort in LED-to-Camera simulation reported in the literature, making CamComSim the first implementation of a LED-to-Camera VLC simulator.

3 System Insight

In this section, we introduce the communication system that CamComSim simulates. We present the testbed used as a reference for the simulator design, give practical insights and propose analytical models that describe the communication properties in LED-to-camera VLC.

3.1 System Description

CamComSim simulates indoor LOS communication between LEDs and cameras with a rolling shutter CMOS sensor. That is, for example, a small and low power color LED and a smartphone. These are respectively the kind of emitter and receiver we use in our testbed. The testbed is similar to the one we used in our experimental work [5]. We implement the VLC emitter on the low-cost STM32L051 microcontroller unit. Following the recommendations of previous studies [1, 3], an OOK modulation scheme with a symbol rate of 8 kHz is used. To avoid any flickering effect, we use the Manchester coding proposed in the literature [1, 3]. The signal is received by the smartphone camera and decoded by an Android application we have developed. We use an LG Nexus 5 smartphone, running the unmodified Android Marshmallow version number 6.0.1, whose 8 megapixels 1080p CMOS sensor can capture up to 30 frames per second. Our Android application sets up the camera parameters to observe the rolling shutter effect produced by the modulating LED, based on the work in [3]. The information is divided and transported in physical layer service data units (PHY-SDU)[1] that have the following structure, depicted in Fig. 1: (1) four synchronization bits to solve the synchronization issue on the receiver part, (2) a payload with a size from 16 to 64 bits, (3) two integrity control bits.

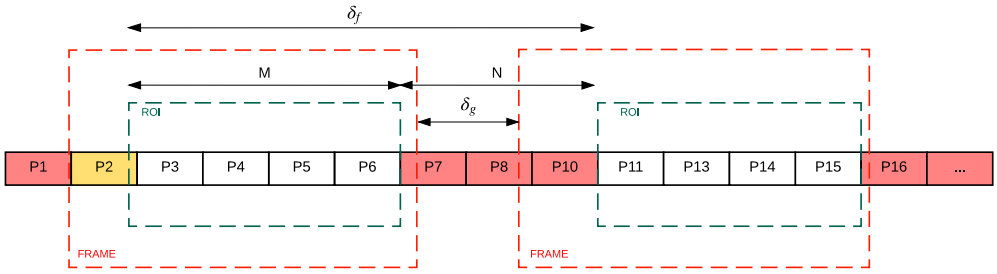


Figure 2: Frame capture time and inter-frame gap, and their relation with the ROI size.

3.2 Analytical model

In a LED-to-camera system, data is received as a series of dark and illuminated stripes in a picture frame captured by the camera. In Fig. 2, we note as f_i the i-th frame captured by the camera and by δ_f the time between the beginning of two consecutive frames. However, data is not continuously received, as a minimum time δ_g exists between two frames. Moreover, as depicted in Fig. 2, the distance d between the LED and the camera also has an impact: when the camera is farther away, the LED transmission is captured for a shorter time, resulting in a smaller region of interest (ROI), i.e. the part of a picture that contains the rolling shutter stripes. The following Sec. 3.3 discusses the impact of d.

We model this unique type of channel using a $M + N$ states Markov modulated Poisson process. Each of these states represents a PHY-SDU reception time slot, i.e. the time duration needed to receive one PHY-SDU.

Practically, the $M+N$ states represent a δ_f time interval and the time slot duration depends on modulation, transmission frequency and PHY-SDU size. The states are divided into two groups: M states corresponding to the camera capture time δ_c, and N states correspond to the inter-frame gap (IFG) δ_g. A Poisson arrival process is associated with each of these $M + N$ states, representing the reception of a packet. In M states, the camera is receiving PHY-SDU, and the arrival rate is $\lambda_1 = (1 - p_e)\lambda_{tx}$, where λ_{tx} represents the transmission rate and p_e is the PHY-SDU error probability (PER). In N states, the camera is not capturing any pictures. Therefore we consider the arrival rate $\lambda_2 = 0$.

3.3 Distance model

As shown previously, the distance between the LED and the camera reduces the ROI size and, as a consequence, cuts down the number of PHY-SDU that the camera can receive per frame, i.e. the M states in Fig. 2.

To include this performance factor into CamComSim, we propose an analytical function that gives the ratio between the ROI and the picture size. In the model discussed in Sec. 3.2, this is the ratio of M states in the $M + N$ states.

We apply photogrammetry rules and give the ROI ratio as a function of the distance d, the LED size l, the camera CMOS sensor size ss and the camera focal distance fc in the following Eq. (1). Note that the ROI is a ratio of the total number of pixels in the picture, i.e. $ROI \in [0, 1]$.

$$ROI = \min\left(1, \frac{l \times fc}{d \times ss}\right) \tag{1}$$

[1]The physical layer service data unit is generally known as a *frame*. To avoid any confusion with the picture frame taken by the camera, we use the term PHY-SDU in this paper.

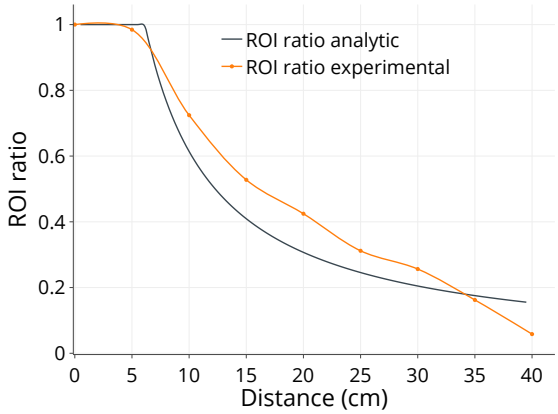


Figure 3: ROI as a function of distance. The orange line shows experimental results, while the green line represents analytical results given by Eq. (1).

To validate the results given by Eq. (1), we measure the ROI experimentally, using the testbed introduced in Sec. 3.1 for distances from 0 to 40 cm. Fig. 3 plots in orange the ROI ratio we observed during our experiments and in green the analytical results computed with the Nexus 5 sensor characteristics: $fc = 35$, $ss = 5.7$ and $l = 10$. This shows that the analytical curve approximates quite well the experimental ROI ratio. However, we notice that the experimental results are better for a distance between 10 and 30 cm but, they become worse than the model at 35 cm. In fact, the light radiance on the camera lens, that our model does not take into account, artificially increases the LED size on the picture when the camera is close to the LED. The difference at larger distance is a consequence of the ambient light which was measured at 650 lux during the experiments, also neglected in Eq. (1).

These two analytical models are the basis of the CamComSim implementation discussed in the following section.

4 Simulator Implementation

4.1 Software architecture

CamComSim is an event-driven LED-to-Camera simulator we developed in Java. Java makes it easy to maintain and distribute code, and it provides built-in multi-platform compatibility for systems with a Java Virtual Machine. Fig. 4 shows the CamComSim software architecture that consists of a simulator kernel class and four core packages.

The *topology* package groups classes that describe the system components: *Led*, *Camera* and *Channel*.

The classes in the *data* package implement the data encapsulation that is done as follow. A *Message* is a set of PHY-SDU that encapsulates a *PhysduPayload*. A *Packet* is a *PhysduPayload* child class, with a sequence number as header and a payload that contains data. Before each simulation, a *Message* is created according to the user settings. The resulting set of *Physdu* is initialized with *Packet* filled with arbitrary data in the payload and a unique sequence number in the header.

The broadcast strategy abstraction is implemented in the *strategy* package. The *Strategy* interface lets the users implement their transmission strategy. This package also contains the straightforward *ReapeatPhysduStrategy* (RP) implementation that consists in repeating each

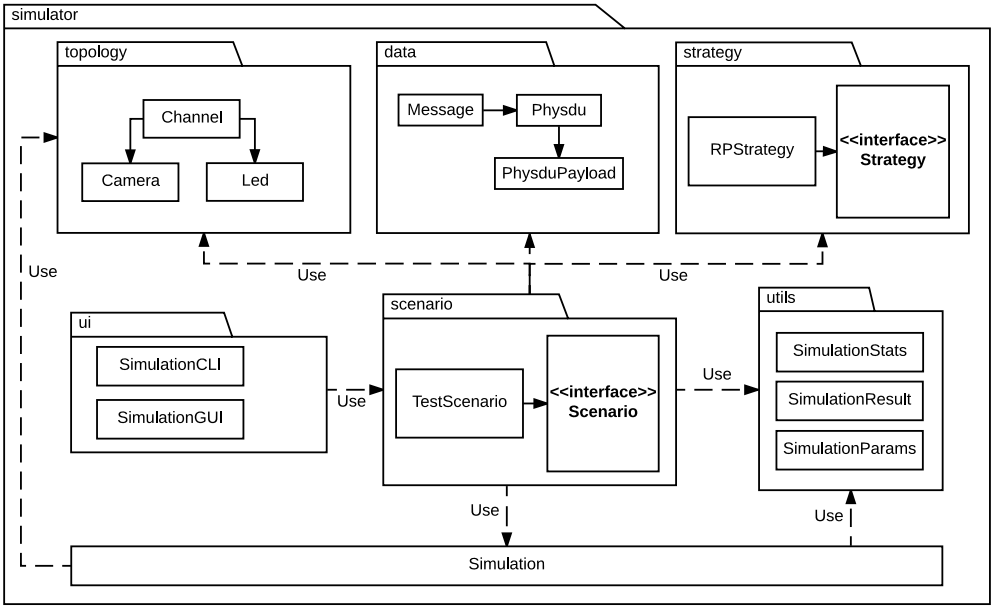


Figure 4: The CamComSim software architecture and packages dependency graph.

PHY-SDU r times, one after the other. When the last PHY-SDU of the message is reached, the process is repeated from the beginning.

Finally the *scenario* package proposes an interface to build a simulation, wiring altogether the *Message*, the *Channel*, the *Led*, the *Camera* and the *Strategy* with the *Simulator* kernel.

Besides, the package *utility* provides helper classes used to compute the simulation results statistics, format and save the results as a JSON file and load or save the simulation parameters. The *ui* package contains a command-line interface (CLI) used to run a simulation scenario.

4.2 Simulator parameters

Our simulator exposes a set of finely grained parameters to describe the LED-to-Camera communication system behavior. Table 1 shows the parameters we use in this paper and expose through CamComSim CLI. As we discuss in Sec. 3, the performance of the LED-to-Camera communication is significantly affected by the distance d, the IFG noted δ_g in Fig. 2, and the LED size l.

Further parameters, introduced in Sec. 3.3, that refer to the CMOS sensor characteristics, are optional but can be considered to refine the channel model, as they impact the ROI. However, smartphone manufacturers rarely provide them. They are the sensor size ss and the focal distance fc.

The PER p_e is the consequence of the errors occurring in a M state when a PHY-SDU is well included in a picture but is wrongly decoded by the smartphone. These errors are bits substitutions induced by interference, low SNR, and artifacts on the picture.

The *PhysduPayload* payload size P and the *PhysduPayload* header size H configure the data encapsulation, as Fig. 1 shows. Given these two settings, the PHY-SDU size is computed as $P + H + SYNC + PB$, where $SYNC$ is the PHY-SDU delimiter symbol, which size is 4 bits, and PB are 2 parity bits. This PHY-SDU size, along with d, l, δ_f and the modulation frequency f, determines the number of the M time slots in Fig. 2.

Param.	Description	Default Value
d	The distance between the camera and the LED (cm)	5
l	The LED size (mm)	4
d_g	The camera inter-frame gap (IFG) ratio	0.1
p_e	The decoder PHY-SDU Error Rate (PER)	0.001
f	The modulation frequency (Hz)	8000
P	The *PhysduPayload* header length (bit)	8
H	The *PhysduPayload* payload length (bit)	16
sc	The scenario implementation class name	*TestScenario*
r	The PHY-SDU repeat number	1
G	The message size (bytes)	50
X	The stop condition (number of PHY-SDU) or stop when received (R)	50000R
i	The number of simulation runs	300

Table 1: Simulator parameters.

Parameter *sc* lets us choose the transmission strategy among the *Strategy* interface implementations. We have implemented and considered only the RP strategy for which the parameters are the size of the message to broadcast, G, the number of consecutive PHY-SDU emissions, r, and the simulation stop condition X. Three stop conditions are defined: (1) a limit on the total number of PHY-SDU sent during the simulation, (2) stopping the simulation when all the PHY-SDU are received (R), and (3) one of the two conditions above. Finally, the CLI parameter i gives the number of simulation runs.

4.3 Kernel Implementation

The CamComSim kernel is implemented in the *Simulator* class. Its role is to produce PHY-SDU emission events (TX) and manage their result.

The number of events, i.e. the number of PHY-SDU sent, is noted c and is updated at runtime. At each clock tick, c is incremented, a TX event is created and processed as follow: (1) the next PHY-SDU in the transmission strategy queue is associated with this event; (2) considering p_e, f, P, H, c, the channel response function gives the event result. The result is one among *reception success*, *reception with errors* or *loss during IFG*; (3) this result is stored in a list to further determine if all the PHY-SDU that form the message are received.

The simulator loops over (1), (2) and (3) until the stop condition X happens, i.e. c has reached the maximum number of PHY-SDU emissions or the complete message is received.

The simulation is repeated i times using the Java *ThreadPoolExecutor* for multi-threading purpose. Finally, the simulations results and statistics are saved in a JSON file for further processing.

5 CamComSim validation

In this section, we present LED-to-Camera simulation results given by CamComSim. To assess the correctness of our simulator, we conduct a series of experiments with the testbed introduced in Sec. 3.1. We set the emitter symbol rate to 8 kHz and place it in standard indoor illumination conditions, near a window and illuminated with neon lights. The illuminance has been measured with a luxmeter at around 650 lux. We compare the testbed performance with the results given by CamComSim for a set of key parameters: the message size G, the number of consecutive PHY-SDU emissions r, the distance d and the PHY-SDU payload length P.

5.1 PHY-SDU Retransmission

To face the IFG bits erasure and ensure that all the packets are well received, [3] proposes to transmit consecutively each PHY-SDU r times in a row. They proved experimentally that for their system, $r = 2$ gives the best performance. We name this strategy RP and implement it in the *ReapeatPhysduStrategy* class.

Fig. 5 shows the goodput at 5 cm for different values of PHY-SDU consecutive retransmissions r, with $G = 50$, $P = 19$, $H = 5$. The stop condition X is set to $50000R$, meaning that the message transmission restarts when each PHY-SDU has been transmitted r times, until the message is received. To avoid infinite loops, we stop the simulation when 50000 PHY-SDU are sent even if the message is not received. In such case, the goodput is 0. The results highlight that the simulation and testbed goodput follow the same tendency when r varies. The best case is when $r = 1$ for which the goodput is 1.6 kbit/s according to CamComSim and 1.7 kbit/s for the testbed, an estimation error of only 6%.

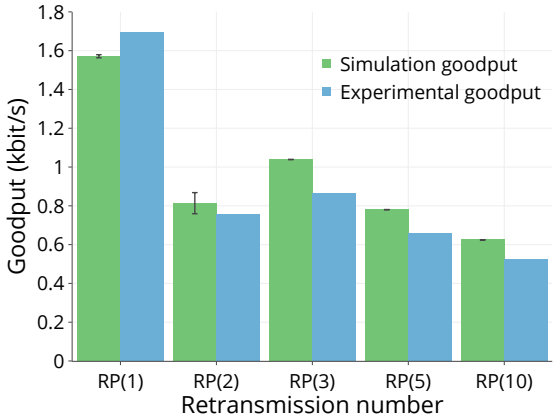


Figure 5: The experimental goodput (blue) compared with the simulation goodput (green) as a function of the number of consecutive PHY-SDU emissions. The bars on top are 95% confidence intervals.

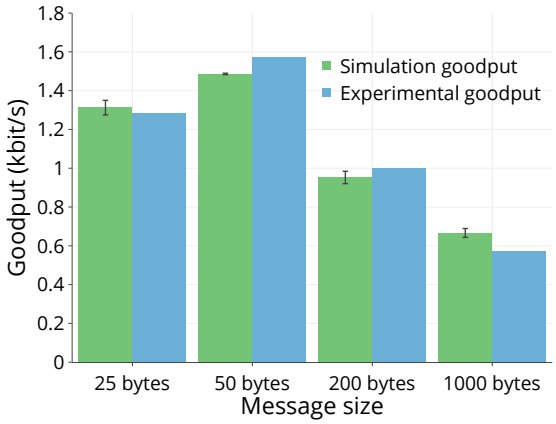


Figure 6: The experimental goodput (blue) compared with the simulation goodput (green) as a function of the distance for different message size (G, bytes).

Based on these results, note that all the simulations that follow use the RP strategy implementation with $r = 1$.

5.2 Message size

We now consider the impact of the message size G on the goodput at a 5 cm distance, with $r = 1$ and a 24 bits length PHY-SDU. Fig. 6 shows that CamComSim results are very close to those of the testbed, confirming that the simulator well considers the impact of the message size. We notice that the goodput reduces when the message size increases, as the RP strategy leads to a large number of useless transmissions: the simulator gives 1.6 kbit/s of goodput for $G = 50$ bytes, while this falls to 670 bit/s when $G = 1000$ bytes. These results differ from the testbed in no more than 7%.

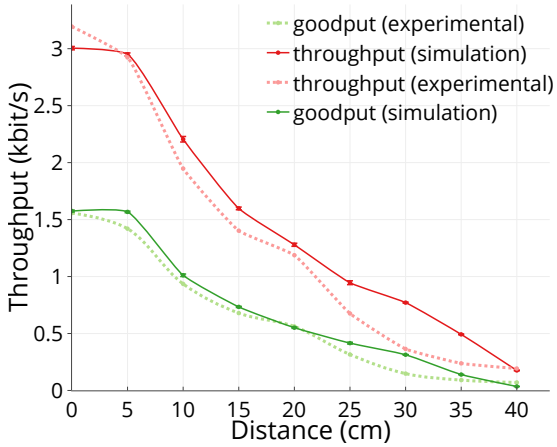


Figure 7: The throughput (red) as a function of the distance, compared with the goodput (green). Dotted-lines show experimental results while plain lines represent simulation results.

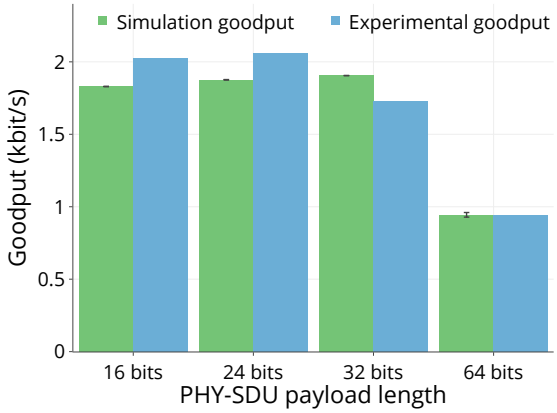


Figure 8: The experimental goodput (blue) compared with the simulation goodput (green) as a function of the distance for different PHY-SDU payload size (bit).

5.3 Distance

Fig. 7 shows the goodput and the throughput as a function of the distance, when the LED broadcasts a 50 bytes message. The PHY-SDU payload is set to 24 bits, with $P = 19$ and $H = 5$. The results show a good match between the simulation and real life results. At 10 cm, CamComSim gives 2.2 kbit/s of throughput when this is 1.94 kbit/s experimentally. The results are closer for the goodput: 0.94 and 1.0 kbit/s respectively for simulation and experimentation, that is only 6% of difference.

5.4 PHY-SDU length

Fig. 8 shows the impact of the PHY-SDU payload size on the goodput at 5 cm, with $G = 50$. The packets are built using the optimal value for P and H, that is to say with just enough bits in the header to label each packet with a unique sequence number.

Both experimental and simulation results show that the optimal PHY-SDU payload size is 24 bits: CamComSim gives a goodput of 1.9 kbit/s while the testbed reaches 2.1 Kbit/s. As Sec 3 explains, large PHY-SDU reduce the encapsulation overhead, but it increases the probability

that the IFG and a small ROI truncate the PHY-SDU.

Fig. 8 brings out that CamComSim well considers this behavior: the goodput becomes 0.9 kbit/s when the PHY-SDU payload size is 64 bits, very close to the experimental results.

This section highlights that CamComSim gives results very close to the testbed for all the parameters we have studied. The difference is around 10% and often less. For all the cases we consider, CamComSim respects the behavior of the LED-to-Camera communication system implemented by the testbed.

6 Use case

In this section, we detail a case study for CamComSim, applied to a real life transmission scenario. Then, we compare simulations with experimental results for the given application. These results will help us design and optimize the broadcast protocol used in the application.

6.1 Use case description

A common issue with cheap consumer electronics is the lack of diagnostics when a dysfunction happens. Manufacturers often blink the state LED with a pattern and color that match with an error code. Such mechanism is easy to implement but leads to inaccurate diagnostics. In such cases, we propose to benefit from this LED to perform LED-to-Camera communication and broadcast a log file that would include helpful information to diagnose a dysfunction. We consider a worst case file size of 1 kbyte that is large enough for events history or debug traces.

6.2 Running the Simulation

As Sec. 4 explains, the simulation has to be described in a class implementing the *Scenario* interface. More precisely, the function `runScenario(SimulationParams params)` that takes as arguments the simulation settings must be implemented.

The implementation must perform the following operations:
(1) Set the transmission strategy to be used during the simulation. That is the RP strategy for this use case. If the simulation aims to evaluate another communication protocol, it must be defined in a class that implements the *Strategy* interface.
(2) Build the *Message* choosing a kind of *PhysduPayload*, for example a *Packet*.
(3) Create the *Led* and the *Camera* objects to build the *Channel* next.
(4) Initialize the *Simulator* kernel with the *Channel*, the *Message*, and the *Strategy*.

After implementing the simulation scenario in CamComSim, we run the simulator with the CLI through the following command: `java -jar camcomsim-1.0.jar -v=1 -i=100 -r=1 -P=14 -H=10 -d=5 -F=8000 -G=1000 -e=0.0001 -X=50000R -dg=0.1 -sc=TestScenario -output=results_output`.
Note that the parameter `-v=1` sets the verbosity level and `-output=results_output` the folder where the results will be saved.

6.3 Results

Fig. 9 compares the goodput given by CamComSim with the goodput that our testbed achieved for the transmission of a 1 kbyte log file as a function of the number of PHY-SDU retransmission r. Note that this is equivalent to $G = 1000$ bytes in Fig. 6. The transmission restarts until the

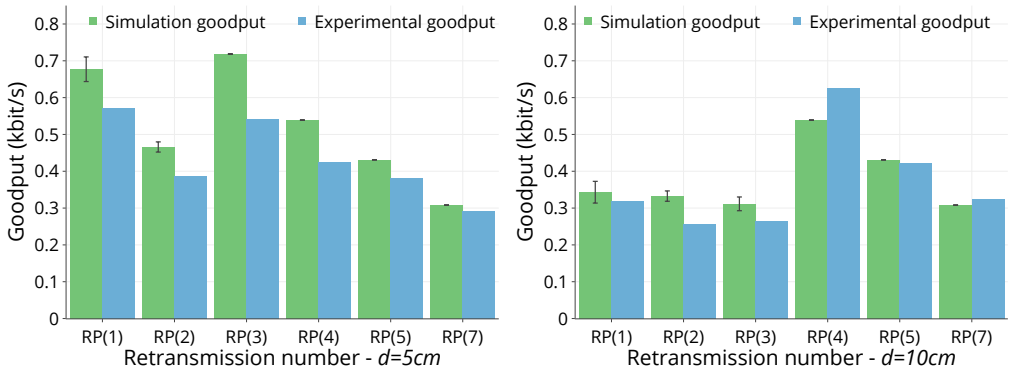


Figure 9: The experimental goodput (blue) compared with the simulation goodput (green) for the use case as a function of the number of consecutive PHY-SDU emission at 5 cm (left) and 10 cm (right).

message is received. The left side plot shows the results when the LED and the smartphone are 5 cm apart, while that is 10 cm on the right side one.

At 5 cm, the simulation brings out that, to obtain the higher goodput, the emitter should send each PHY-SDU one or three times consecutively, i.e. $r = 1$ or $r = 3$. The goodput is respectively 680 and 720 bit/s in these cases. This finding is the same on the testbed for which the goodput is 570 bit/s when $r = 1$ and 540 bit/s when $r = 3$.

Because the ROI decreases with the distance, the behavior is different when the smartphone is 10 cm far from the LED. In this situation, $r = 4$ stands out clearly to be the best choice both for the simulation and the experiments. The goodput then becomes 620 bit/s on the testbed and 540 bit/s with CamComSim.

Since the results are very close to the reality, using CamComSim highly reduces the search space for the experimental optimization of a system. As shown by these results, the best value for r can be decided using only simulations.

7 Conclusion

In this paper, we have introduced CamComSim, the first simulator for the design, the prototyping and the development of protocols and appliations for LED-to-camera communication. Our event driven simulator relies on a standalone Java application that is easily extensible through a set of interfaces. We have validated CamComSim comparing simulation results with the performance reached by a real life testbed. Then, we illustrated with a practical use case the complete usage of CamComSim to tune a broadcast protocol that implements the transmission of a 1 kbyte log file. The results highlight that our simulator is very precise and can predict the performance of a LED-to-Camera system with less than 10% of error in most cases. The availability of very accurate simulators offers a great ease of use and the opportunity to tune protocols without the burden of realizing experiments on a testbed.

We are now interested in the implementation of additional PHY layers proposed in the literature. That includes FSK modulation, line codes as 4B6B or 8B10B, and the study of the symbol rate influence on the PER. We hope CamComSim will help researchers evaluate new communication protocols or coding techniques. CamComSim is available at `http://vlc.project.citi-lab.fr/camcomsim`.

References

[1] C. Danakis *et al.*, "Using a CMOS camera sensor for visible light communication," *Proc. IEEE Globecom Workshops*, 2012.

[2] IEEE 802.15 WPAN 15.7r1 Amendment Study Group (2015). [Online]. Available: http://www.ieee802.org/15/pub/IEEE%20802_15%20WPAN%2015_7%20Revision1%20Task%20Group.htm

[3] J. Ferrandiz-Lahuerta, D. Camps-Mur, and J. Paradells-Aspas, "A Reliable Asynchronous Protocol for VLC Communications Based on the Rolling Shutter Effect," in *Proc. IEEE GLOBECOM, 2015.*

[4] H.-Y. Lee *et al.*, "RollingLight," in *Proc. ACM MobiSys*, 2015.

[5] A. Duque *et al.*, "Unleashing the power of LED-to-camera communications for IoT devices," in *Proc. ACM VLCS Workshop*, 2016.

[6] M. Sharif and A. Sadeghi-Niaraki, "Ubiquitous sensor network simulation and emulation environments: A survey," *Journal of Network and Computer Applications*, vol. 93, pp. 150–181, 2017.

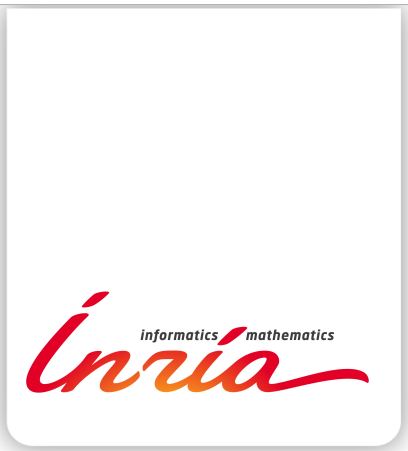
[7] D. Tagliaferri and C. Capsoni, "Development and testing of an indoor VLC simulator," in *Proc. IEEE IWOW*, 2015.

[8] A. Aldalbahi *et al.*, "Extending ns3 to simulate visible light communication at network-level," in *Proc. IEEE ICT*, 2016.

[9] C. Ley-Bosch *et al.*, "Implementing an IEEE802.15.7 Physical Layer Simulation Model with OMNET++," in *Proc. DCAI*, Springer, 2015.

[10] A. Musa, M. D. Baba, and H. M. A. Haji Mansor, "The design and implementation of IEEE 802.15.7 module with ns-2 simulator," in *Proc. IEEE I4CT*, 2014.

Contents



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399